5-1-2019

Support for User Generated Evolutions of Goal Models

Boyue Caroline Hu
University of Toronto

Alicia M. Grubb
Smith College, amgrubb@smith.edu

Support for User Generated Evolutions of Goal Models

Boyue Caroline Hu
Department of Computer Science
University of Toronto, Toronto, Canada
boyue.hu@mail.utoronto.ca

Alicia M. Grubb
Department of Computer Science
Smith College, Northampton, MA, USA
amgrubb@smith.edu

***Abstract*—Goal models are used in early phase requirements engineering to elicit stakeholders' intentions, analyze dependencies, and help stakeholders make trade-off decisions about the project and its interaction with the environment. The Evolving Intentions framework extended goal model analysis to evaluate how models change over time, by creating simulation paths showing possible evolutions of the model. More recently, we extended this analysis to allow users to explore states along the path and generate their own simulation paths. However, this approach is limited by users' ability to comprehend the state space, which grows exponentially with the size of the model. In this paper, we explore using *filters* to reduce the number of viewable solutions enabling users to create their own simulation results. We present our approach and initial validation, including an analysis of prior models and a review of expert feedback.**

I. Introduction

Goal-Oriented Requirements Engineering has been advocated to help stakeholders make trade-off decisions at an early stage. Goal models illustrate intentions, requirements and constraints to help stakeholders understand and evaluate potential project scenarios and facilitate early decision making [1][2].

The Evolving Intentions framework allows stakeholders to specify changes in their intentions and the project domain over time and ask questions about a project's evolution [3]. BloomingLeaf is a web-based tool that implements the Evolving Intentions framework and provides formal automated analysis of goal models [4]. BloomingLeaf uses the satisfaction values and evolutionary functions assigned to the intentions in the model, and produces a *simulation path* to show how the fulfillment of each intention in the model changes over time.

A single simulation path might not be sufficient to understand the domain. Stakeholders may want to create their own paths that are different than the automatically generated one, in order to gain more insights about the evolution of the project. Thus, we added additional functionality to BloomingLeaf that allows users to navigate to a state in the simulation path and find all possible *next states* in the path [5]. As the model becomes bigger, the state space of this problem (i.e., number of possible next states) grows exponentially in the worst case. Presenting all next states to users makes it difficult for them to review and customize their simulation path. This problem of choice selection has already been well studied by researchers in psychology [6][7], where they found that unlike computers, humans are less effective at making choices as the number of options increases. The aim of this research project is to improve the usability of our user generated simulation path feature by reducing the number of next states presented to users at each step.

We propose using *filters* to reduce the number of solutions for each next state in the path. Due to the nature of goal models as well as the *fulfillment data* involved in choosing a next path, selecting appropriate filters is a nontrivial task. In this paper, we present our approach to filtering next state results. Using a student applying to graduate school as a motivating example, we present an overview of the problem of choosing a next state within the context of simulation in the Evolving Intentions framework, modeled in Tropos [8][9]. We present our proposed filters based on the metrics in the goal modeling literature [10]. We consider the effectiveness of these filters using eight models from the literature and unstructured interviews with experts. We explore two research questions: (RQ1) To what extent does the filters approach reduce computation time and the number of returned states? (RQ2) To what extent do experts find this approach helpful?

The remainder of this paper is organized as follows. Sec. II introduces our motivating example and relevant background. Sec. III describes our extension to BloomingLeaf. Sec. IV presents our initial validation to inform our research questions. We compare our work with other analysis techniques in Sec. V and conclude in Sec. VI.

II. Background

In this section, we introduce goal modeling and the Evolving Intentions framework using our motivating example.

Graduate School Application (GRAD). Consider an undergraduate student interested in graduate school. The student wants to make sure he satisfies all the requirements and doesn't miss any opportunities. He needs to consider the order of satisfying tasks (i.e., fulfilling requirements). One key decision in this process is in which order should these tasks be completed. For example, should Do Research Work be completed before or after Internship. Fig. 1 shows the partial goal model of this decision, consisting of the intentions of the Student and his Recommenders (i.e., actors). These intentions (i.e., goals, tasks, and soft goals) are connected by contribution links and decomposition links (see legend in Fig. 1). The Student's root-goal Be Admitted to Graduate School is and

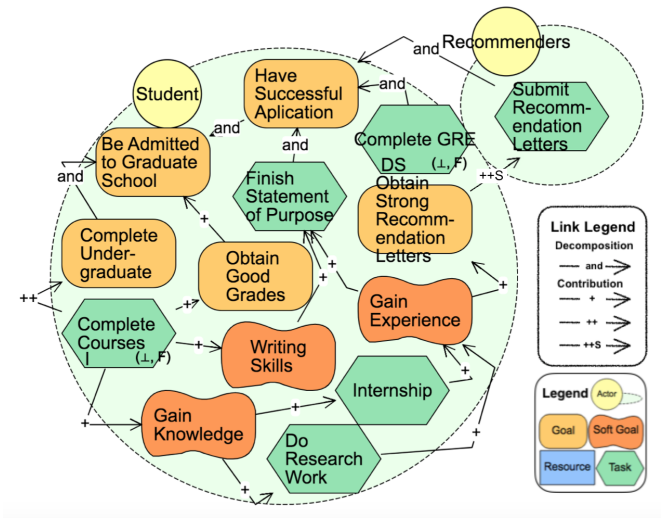


Fig. 1: GRAD Model: A goal model of a student's graduate school application.

decomposed into the goals Complete Undergraduate and Have Successful Application, meaning that Be Admitted to Graduate School is satisfied only if both Complete Undergraduate and Have Successful Application are satisfied. Have Successful Application is then AND decomposed by tasks Finish Statement of Purpose, Complete GRE, and Submit Recommendation Letters, meaning that all three must be fulfilled to satisfy Have Successful Application. All of the other links in Fig. 1 are contribution links (see [8] for a full list of contribution types). The + [resp. ++] link propagates some [resp. full] evidence from the source to the target intention. The ++S link means that only positive evidence is propagated to the target intention.

Evidence Pairs. When evaluating a goal model, we assign *evidence pairs* (s, d) labels to intentions, where $s \in \{F, P, \bot\}$ is the evidence *for* (i.e., satisfaction) and $d \in \{F, P, \bot\}$ is the evidence *against* (i.e., denial) the fulfillment of an intention [8]. F [resp. P] means there is full [resp. partial] evidence for or against the fulfillment of an intention, and $\bot$ represents null evidence. For example, in Fig. 1, Complete Courses and Complete GRE have been assigned $(\bot, F)$ in the initial state, meaning there is full evidence against their fulfillment (i.e., neither task is completed).

An intention can be assigned one of nine possible evidence pairs, by taking the product of the sets s and d, resulting in the following partial order from most satisfied to most denied: $(F, \bot)$; $(P, \bot)$, (F, P); $(\bot, \bot)$, (P, P), (F, F); $(\bot, P)$, (P, F); and $(\bot, F)$. Evidence pairs can be assigned by the modeler or as a result of propagation-based analysis from another intention (via the links discussed above). See [11] for a full set of propagation rules used in the Evolving Intentions framework.

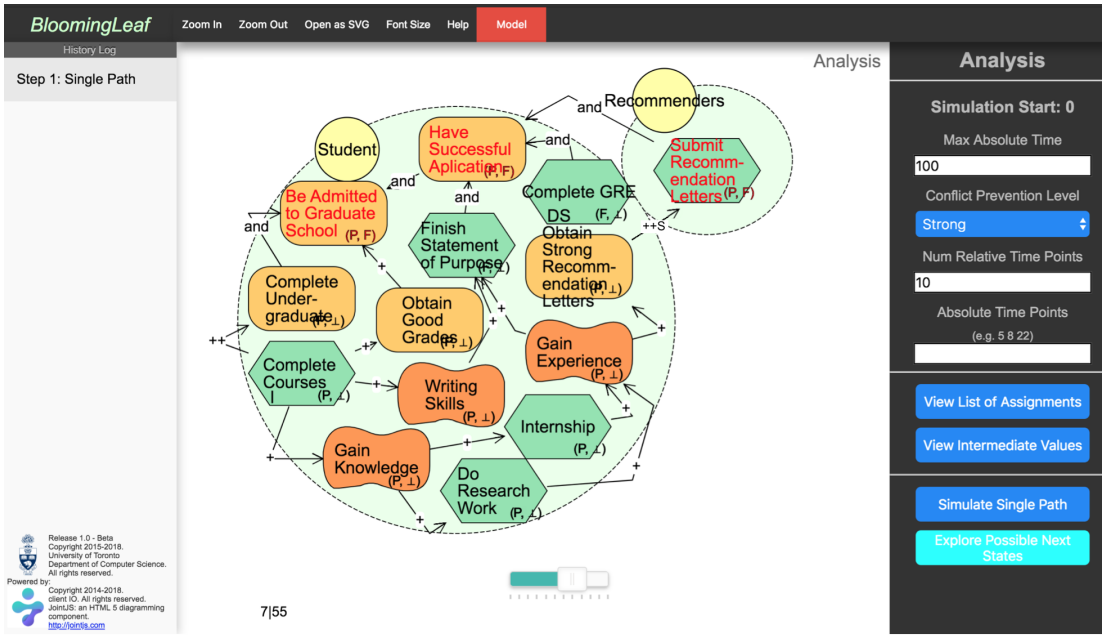
Evolving Intentions. As introduced in Sec. I, the Evolving Intentions framework allows the fulfillment of intentions, specified with evidence pairs, to change over time. We define four functions to describe how the fulfillment of an intention changes over a single time interval: CONSTANT, INCREASE, DECREASE, and STOCHASTIC [11]. Functions indicate the desired evolution of intentions, and are assigned prior to analysis. For example, in the GRAD model the Complete Courses task is assigned an INCREASE function (visually represented by the I on Complete Courses in Fig. 1). With an initial value of $(\bot, F)$, this means that over time, there will be increased evidence for and decreased evidence against the fulfillment of Complete Courses, along the partial order of evidence pairs. In the GRAD example, Complete GRE also changes over time, and we define it using a step-wise function over two intervals, where within each interval the value remains CONSTANT. Complete GRE has the value $(\bot, F)$ in the first interval, and $(F, \bot)$ in the second interval. This pattern, known as DENIED-SATISFIED in the framework [11], is primarily used to model tasks that will be completed in the future. When these evolving functions are combined with other goal model information, we can simulate possible evolutions of the goal model.

BloomingLeaf Analysis. BloomingLeaf is a web-based tool for modeling and analyzing goal models [4]. Stakeholders first create a goal model and add appropriate evolving functions before completing any analysis. The Analysis View, shown in Fig. 2a, lists two different types of analysis: Simulate Single Path and Explore Possible Next States.

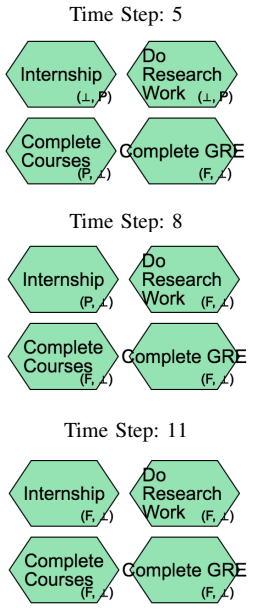
Simulate Single Path presents the user with one possible evolution of their model over a pre-specified number of time points. For example, the screenshot in Fig. 2a shows a single result of Simulate Single Path for the GRAD model at time point seven. The full solution contains eleven time points and the user can switch between time points using the slider below the model. Fig. 2b illustrates other portions of this simulation path for the tasks discussed above, focusing on the time points (i.e., time points 5, 8, and 11) where the evaluations of these tasks change. We omit the full model at each time point in the path for brevity.

This simulation result also illustrates the evolving functions we introduced above. Recall that Complete GRE was initially denied $(\bot, F)$ (see Fig. 1) and as a result of the DENIED-SATISFIED function it is satisfied $(F, \bot)$ at time point five (see Fig. 2b). Complete Courses was assigned an INCREASE function with an initial value of denied $(\bot, F)$ (see Fig. 1). In Fig. 2b, we see the evaluation of Complete Courses becomes more fulfilled; as it is partially satisfied $(P, \bot)$ at time point five and then satisfied $(F, \bot)$ at time point eight.

Overall, the simulation illustrated in Fig. 2 gives the result that the student can satisfy Have Successful Application by satisfying $(F, \bot)$ Complete Courses and Do Research Work at time point eight and then satisfying $(F, \bot)$ Internship at time point eleven (see Fig. 2b). However, the student wants to further explore the question: "What would happen if he completes an internship next (by satisfying Internship), rather than doing research?" The Explore Possible Next States analysis, allows users to step into any time point in the single path solution and visualize all the possible next states. For



(a) Screenshot of BloomingLeaf's Analysis view showing a result of Simulate Single Path for the GRAD model at time point seven.



(b) Selected tasks at time points 5, 8, and 11.

Fig. 2: Simulate Single Path result for the GRAD example.

example, when the student clicks Explore Possible Next States from time point seven (as in Fig. 2a), the Next States pop-up window is invoked displaying the first possible next state with an indicator at the top displaying that there are 640 possible states for the next time point. We exclude this initial pop-up for space considerations, but it looks similar to Fig. 3a where we have already applied our filters.

In BloomingLeaf, both Simulate Single Path[1] and Explore Possible Next States are encoded as a *constraint satisfaction problem* (CSP). The number of possible next states for the GRAD model, with only 14 intentions, is in the hundreds. This is because CSP searches for all solutions that satisfy the constraints in the model, and with very few constraints[2], the solution space grows exponentially D^n (i.e., *state explosion problem* [12]), where D is the number of possible evidence pairs and n is the number of intentions. While actual runtimes for Simulate Single Path have been shown to be appropriate [3], when we return the entire state space for a single time point (as in Explore Possible Next States), this presents a significant burden to the user as the number of returned states grows. Our work aims to improve the interpretation of these results for users.

III. Extension Overview

As introduced in Sec. II, we allow users to create their own simulation paths by exploring all possible next states iteratively through the solution space. This creates an increased cognitive burden on users as the size of the model grows. We assist users in selecting next states by introducing two approaches to filtering the number of next states shown to the users in BloomingLeaf (i.e., domain and solution reduction).

Domain Reduction. First, we look at adding additional constraints to the CSP (i.e., domain reduction). We reduce the domain of each Explore Possible Next States search by allowing users to filter out *conflicting* and *null evidence* values. As described in Sec. II there are nine possible evidence pairs and of these four are conflicting values: (F, P), (P, P), (F, F), and (P, F). Prior work and BloomingLeaf allows users to prevent *Strong*, *Medium*, and *Weak* conflict values when using Simulate Single Path [4] [8]. We adapt conflict avoidance as a filter to be applied to the Explore Possible Next States feature. This means stakeholders can find all the next states that do not contain any conflicting evidence pairs. Similarly, we allow users to filter out null evidence values $(\perp, \perp)$ or *None*.

In Fig. 2a of the GRAD example, the student is exploring the next state from time point seven. Notice that the original simulation had Strong as the Conflict Prevention Level, meaning that the student does not want to include the strong conflict values, i.e. (F, F), in the single path. In Sec. IV, we show how these filters reduce computation times for next state results.

Solution Reduction. Second, we introduce solution filters to exclude possible states reducing the number of total next states shown to the user. CSP solvers treat all valid states equally. For example, a state with all the intentions fulfilled and another state with no intentions fulfilled differ in context to users but are not distinguished by the solver. Stakeholders have partial or full contextual information to favour some

[1]Called *Simulation over All Evolving Intentions* in [11].

[2]The state space can be reduced by adding additional evolving functions.

TABLE I: List of filters used in the Next States analysis.

Domain Reduction Filters		
Name	Description	Example Usage
Remove Conflict Values	Remove all solutions that contain conflict values for any of the intentions: (F,P), (P,P), (F,F), and (P,F).	If the user is looking for absolute satisfaction or denial of the intentions in the model, *Remove Conflict Values* will eliminate all solutions that present evidence of both.
Remove No Information	Remove all solutions that contain the $(\perp,\perp)$ evidence pair.	It the user want some evidence for or against each intention, where no information would not help in the decision.
Solution Reduction Filters		
Name	Description	Example Usage
Least/Most Tasks Satisfied	Keep only the solutions with the least/most number of tasks with the evaluation label satisfied $(F,\perp)$.	In the GRAD example, if the student is looking for the minimum number of tasks he needs to complete to be admitted to graduate school.
Least/Most Goals Satisfied	Keep only the solutions with the least/most number of goals with the evaluation label satisfied $(F,\perp)$.	This would be useful for the student in the GRAD example to view the worst case and best case scenario.
Least/Most Resources Satisfied	Keep only the solutions with the least/most number of resources with the evaluation label satisfied $(F,\perp)$.	Consider a business person making budgets of all the resources he needs, *Least Resources Needed* would give a lower bound estimation and *Most Resources Needed* would give an upper bound.
Least/Most Actors Involved	Keep only the solutions with the least/most number of actors involved. An actor is involved when at least one of their intentions is satisfied.	In the GRAD example, if the student were to ask whether he can finish the entire application process all by himself.
Satisfaction of the Most Constrained Goal	Keep only the solutions with the status of the most constrained goal being satisfied. Most constrained goals are goals with the smallest domain in the model.	This usually helps when users want to explore the satisfiability of some or all goals in the model.

states over others, which can make user generated simulations feel more realistic for stakeholders. Based on the research on goal model metrics [13], we define five filters over the solution space and list them in Tbl. I. Most filters are based on an attribute, allowing for the maximization or minimizations of the characteristic. For example, the first row of Solution Reduction Filters in Tbl. I lists *Least/Most Tasks Satisfied*. Most Tasks Satisfied displays any permutation of states where the maximum number of tasks are satisfied. In the GRAD example (see Fig. 2a), the most number of tasks that can be satisfied in the next state is 3 so all states with 3 tasks satisfied will be shown. These filters reduce the number of solutions shown to users helping them to pick a next state. In Sec. IV, we demonstrate to what degree the number of states is reduced by applying each of these filters.

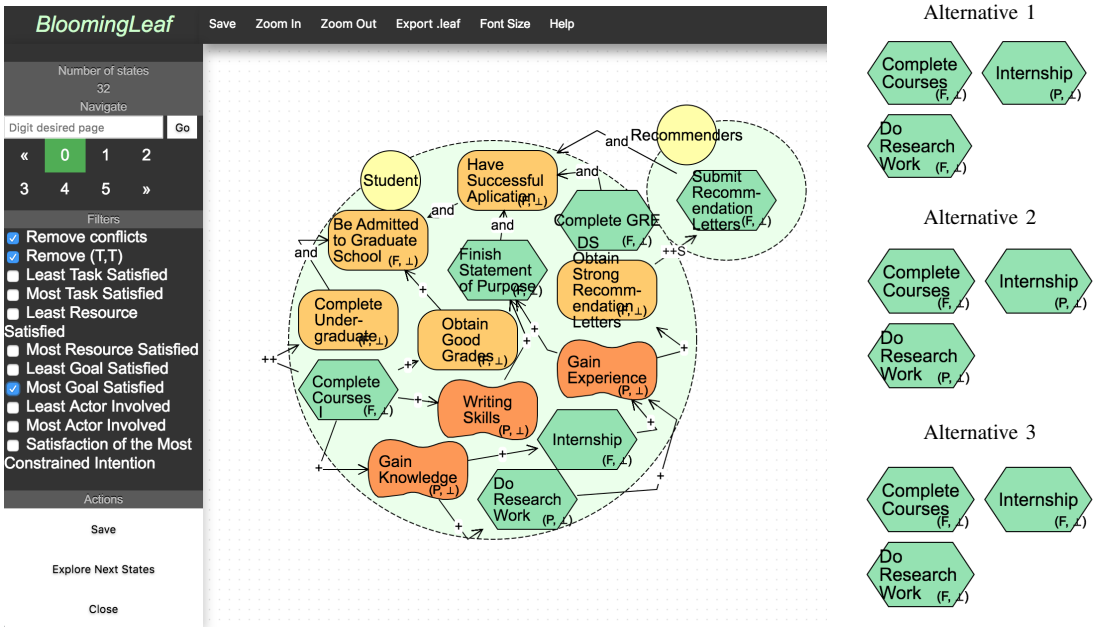
Implementation in BloomingLeaf. In BloomingLeaf, once the user has selected Explore Possible Next States and are viewing the Next States pop-up window (see Fig. 3a), they can complete the following actions:

- Browse through all next states. Users can switch between states by selecting the indices at the top of the left panel, or by entering a desired state number and clicking Go.
- Applying a filter by clicking on the check box next to the filter name. For example, in Fig. 3a Most Goal Satisfied has been applied.
- The Save button records the selected state and generates the remainder of the path in the original analysis window.
- The Explore Next States button records the selected state and then in the same window generates all states for the following time point. This allows users to incrementally generate next states until they have completed the path.

Demonstration of Filters on the GRAD example. As we introduced in Sec. II, the student, in the GRAD example, is exploring the question: "Should Do Research Work or Internship be satisfied first?"

Instead of continuing to generate new random paths, the student decides to pick a next state himself to customize this path. Using BloomingLeaf, the student selected time point seven (see Fig. 2a), where Internship and Do Research Work were partially satisfied $(P,\perp)$, and clicked Explore Possible Next States. This resulted in over 600 states for the next time point, which is unrealistic to review. Thus, the student applies filters to make this task more manageable (as shown in Fig. 3a). He first applies Remove Conflict and Remove $(\perp,\perp)$ to find a state without any conflict value or no information, reducing the number of states to 64. He then selects Most Goals Satisfied to find the states where some of his goals are satisfied. This results in 32 states, as shown in Fig. 3a. He reviews some of the 32 states and selects one where Internship is fulfilled and Do Research Work is not. Using this new state for time point eight, the student generates the remainder of the path, where Be Admitted to Graduate School is eventually satisfied. The student realizes that both options are valid and decides to complete an internship before completing research.

To illustrate selecting a next state, fragments of some alternative next states are demonstrated in Fig. 3b. Do Research Work would still be satisfied before Internship if Alternative 1 is selected. Alternative 2 and 3 do not indicate any order of the two tasks being satisfied. As a result, all three alternatives are not the desired next state in this example.



(a) Screenshot of the Next States pop-up window in BloomingLeaf, showing the GRAD model on the centre canvas with three filters selected.

(b) Alternative Next States for Time Point 8

Fig. 3: Demonstration of Next State analysis in BloomingLeaf with filters.

IV. Validation

In this section, we describe our initial validation of our extension and explore two research questions: (RQ1) To what extent does the filters approach reduce computation time and the number of returned states? (RQ2) To what extent do experts find this approach helpful? We define *computation time* as the time required for our CSP solver to find a solution. To explore these research questions, we analyze effectiveness by measuring the changes in runtimes and reductions in the number of solutions presented to users, and we analyze usability by gathering expert opinions on the approach.

Effectiveness Evaluation. As shown in the GRAD example, the number of states returned to the student was greatly reduced from over 600 to 32 states by applying the filters. To ensure that these results were not unique to the GRAD example, we investigate RQ1, how each filter reduces the number of returned states and computation time. We selected eight models from the goal modeling literature of varying sizes (i.e., number of intentions and links). We added evolving functions to the Scheduler model, as it was an untimed model. For each model, we used Simulate Single Path to create paths with at least ten time points and then selected Explore Possible Next States from time point four. Since path lengths varied by model, we chose time point four prior to analysis because a next state of time point five would be at most in the mid-point in the path. Tbl. II lists the results of our analysis with each column representing a single model. The first two rows list attributes of the model. The middle block lists the initial number of returned states for selecting Explore Possible Next States at time point four and the number of solutions returned after applying each filter independently. For example, the GRAD model is shown in the first column of Tbl. II. We can see that by selecting *Remove All Conflict Values*, the number of returned states is reduced from 711 to 64.

Based on the results in Tbl. II, without preferences most models resulted in hundreds of states and applying filters resulted in a reduction of the number of states returned to the user. We examined the cases where filters did not reduce the number of states and found that the contents of the model caused these anomalies. For example, in the GRAD model *Least/Most Resources Needed* did not impact the number of states because there is no resources in the GRAD model. Also, *Remove All Conflict Values* resulted in zero states in the Bike Lanes Full and the Spadina Plan models because there were intentions connected by both + and - links, resulting in conflict.

The bottom section of Tbl. II lists the computation times for Explore Possible Next States with and without removing conflict or none values. For example, the computation time of the Scheduler model was reduced from 317 ms to 187 ms [resp. 173 ms] by removing conflict [resp. none] values. We were not able to collect runtime data from all models because removing all conflict or none values resulted in an over-constrained model with no valid solution. Apart from the results for the Bike Lanes Full model, these measured reductions would not be observable by users; thus, we will focus future effort on reducing the number of solutions. We conclude that the applicability of each filter varied based on the model structure, but overall applying solution reduction

TABLE II: Results of the Effectiveness Study: Measurements of the computation times, and the number of solutions shown to users, as the result of applying each of our proposed filters over eight selected models, of varying sizes.

Model Name	*GRAD*	*BLE [11]*	*WME [11]*	*Scheduler [14]*	*Spadina Plan*	*Spadina Opp*	*Spadina Pro*	*Bike Lanes Full*
Num. Intentions	14	8	20	18	43	38	28	30
Num. Links	18	7	16	20	55	35	31	37
Measurement	Number of Solutions							
Explore Possible Next States - No Pref.	771	486	3882	6369	20	5832	36	51152
Remove All Conflict Values	64	110	30	1173	0	200	8	0
Remove No None	512	0	3056	4093	16	4608	32	45056
Most tasks satisfied	32	486	96	288	20	972	36	51152
Least tasks satisfied	66	486	264	105	20	1944	36	51152
Most actors involved	514	468	3456	504	20	3888	18	51152
Least actors involved	257	18	426	105	20	1944	3	51152
Most resources needed	771	486	3882	4536	20	5832	36	51152
Least resources needed	771	486	3882	1833	20	5832	36	51152
Most goals satisfied	192	2	3882	504	10	24	36	68
Least goals satisfied	195	352	3882	713	10	3072	36	3448
Satisfaction of the most constrained goal	771	486	3882	6369	0	5832	36	51152
Measurement	Computation Time in milliseconds							
Explore Possible Next States - No Pref.	131	119	286	317	88	388	96	4141
Remove All Conflict Values	121	47	N/A	187	N/A	N/A	92	N/A
Remove No None	97	N/A	226	173	86	N/A	89	1444

N/A indicates that no measurement was collected because the model was over-constrained.

filters reduced the number of returned states, answering RQ1.

Expert Analysis. Normally, we would investigate RQ2 in the context of a controlled experiment with a significant number of participants. Since our implementation is still a "proof of concept" any results would be greatly confounded by our interface. Instead, we gathered expert feedback about our extension and implementation, by walking volunteers through a mock trial of our proposed study treatment. We solicited expert feedback of graduate students and professors at the University of Toronto through the Software Engineering group mailing list. We defined an expert as someone who has conducted research in goal modeling or formal reasoning techniques. Five experts volunteered to review our tool and technique and we interviewed them individually. We gave each expert the GRAD model and asked them to answer the question: "For this student to be successfully admitted to graduate school, in which order should the five tasks be completed?" After generating a path and selecting Explore Possible Next States, we asked them to select a next state without and then with filters. We timed how long it took them to find an answer and asked them to comment on where they spent the most time.

The experts took between two and six minutes to complete this task and selected between three and six filters. Anecdotally, most of the experts time was spent on deciding which filters to apply. Two of the experts commented that the names of the filters were not self-explanatory so it was not clear which filters would help solve the problem. All experts agreed that the initial number of states returned was too many to consider. One indicated that it would be much easier for them to re-build the model and add more constraints to it in order to reduce the number of solutions. Another expressed concerns that users might not have sufficient RAM to process these large problems. As a result, the participants only did the task with filters as requested by them. One suggested that it would be easier for the users if we can filter based on the status of a specific intention. Finally, all of the experts agreed that the filters saved them time and effort.

These interviews were not intended to provide conclusive evidence and have obvious *threats to validity* (e.g., experimenter expectancies, low sample size). Instead, they were intended to inform the design of our future experimentation. Thus, these results are far from conclusive in answering RQ2, but it has strengthened our hypothesis that users will find filters useful. We need to make significant improvements in BloomingLeaf to help users select appropriate filters.

V. Related Work

We are not the first to look at optimizing goal model analysis and results. This work builds on the prior goal model analysis work of Giorgini and collaborators [8][9] and extends the Evolving Intentions framework [3][11]. Similarly, Aprajita et al. worked on visualizing how intentions change over time [15]. Our initial set of solution reduction filters was loosely based on the work of Franch et al. [13]. They created metrics over goal models and hoped to create additional metrics to assist the needs of goal model analysis.

Matthew et al. investigated exploiting the "key" decisions within goal models to improve the efficiency of propagation-based analysis results for very large goal models [16]. As discussed in Sec. IV, one of our experts recommended applying filters based on the status of specific intentions. Future

work could incorporate this idea of "key" decisions. Nguyen et al. proposed finding incremental solutions in goal model analysis based on changes in intentions and past results [17]. Future work could incorporate these optimization functions. Although our domain reduction filters refine the CSP, they do not result from any changes in the intentions.

Others have investigated preferences in goal models. Liaskos et al. allowed users to indicate preferences between mandatory and optional goals, giving priorities to some goals [18]. By indicating preferences, their work assisted analysis algorithms to efficiently search for solutions. Jureta et al. also considered preferences in goal models, by describing systems-to-be in terms of candidate solutions and optional requirements to be used as criteria for comparison in analysis [19]. Some may view our work as allowing the user to state preferences of evidence pairs in analysis results.

The concept of filters was also used in prior work in the domain specific modelling literature. Zarrin et al. used filter, together with map and reduce, as lists of operations within a function to specify domain specific languages [20]. Melnik et al. defined a similar concept in order to use semantics to guide the implementation for particular schema definition languages and mapping languages for model operators (e.g., Compose, Extract and Merge) [21]. Our work differs in that we are not applying these filters directly to the structure of the model, but filtering possible evaluations of the model.

VI. Conclusion & Future Work

In this paper, we introduced using filters to reduce the state space of our Explore Possible Next States analysis, improving users' ability to create their own simulation results. Using the motivating example of a student desiring to be accepted into graduate school, we described filters for domain and solution reduction, and our implementation in BloomingLeaf. As a preliminary validation of our filters, we measured changes in the computation time and the number of returned states across a variety of models, and interviewed experts in order to gain insights into how we can improve our analysis. We found that applying solution reduction filters reduced the number of returned states in RQ1, and found some evidence that experts found filters helpful in RQ2.

In future work, we will investigate using stakeholders stated or implicit preferences, over their goal models, for the purpose of creating context dependent filters and guiding users in selecting the most appropriate filters. We measured the solution reduction filters in isolation. Going forward, we will consider interactions between filters and evaluate the effectiveness of combining filters. We will complete a more extensive study for the Solution Reduction filters, by measuring reductions across multiple paths and at multiple time points. As part of our Next State analysis, we will explore allowing users to update the evidence pair assignments for each intention in the model, which will allow us to prune the solution space using these new values. Finally, in future work we will empirically validate the effectiveness of our approach with goal model users.

VII. Acknowledgement

We thank Marsha Chechik and the members of the Software Engineering group at the University of Toronto for their assistance. Boyue Caroline Hu completed this research with the support of a University of Toronto Excellence Award.

References

[1] J. Horkoff, T. Li, F.-L. Li, M. Salnitri, E. Cardoso, P. Giorgini, J. Mylopoulos, and J. Pimentel, "Taking Goal Models Downstream: A Systematic Roadmap," in *Proc. of RCIS'14*, May 2014, pp. 1–12.

[2] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, J. Mylopoulos, and P. Giorgini, "Goal-Oriented Requirements Engineering: A Systematic Literature Map," in *Proc. of RE'16*, 2016, pp. 106–115.

[3] A. M. Grubb and M. Chechik, "Looking into the Crystal Ball: Requirements Evolution over Time," in *Proc. of RE'16*, 2016, pp. 86–95.

[4] ——, "BloomingLeaf: A Formal Tool for Requirements Evolution over Time," in *Proc. of RE'18: Posters & Tool Demos*, 2018, pp. 490–491.

[5] B. C. Hu, A. M. Grubb, and M. Chechik, "Exploring Next States and Alternative Paths in Goal Model Analysis," *Review of Undergraduate Computer Science, University of Toronto*, 2019, forthcoming.

[6] B. Schwartz, *The Paradox of Choice: Why More Is Less*. Harper Collins, 2003.

[7] S. Iyengar, *The Art of Choosing*. Little, Brown Book Group, 2010.

[8] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Formal Reasoning Techniques for Goal Models," *J. on Data Semantics*, vol. 1, pp. 1–20, 2003.

[9] R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Simple and Minimum-Cost Satisfiability for Goal Models," in *Proc. of CaiSE'04*, 2004, pp. 20–35.

[10] X. Franch, "A Method for the Definition of Metrics over i* Models," in *Proc. of CAiSE'09*, 2009, pp. 201–215.

[11] A. M. Grubb, "Modeling and Analyzing the Evolution of Requirement over Time using Goal Models," Ph.D. dissertation, University of Toronto, 2019.

[12] E. M. Clarke, W. Klieber, M. Novacek, and P. Zuliani, "Model Checking and the State Explosion Problem," in *LASER 2011: Tools for Practical Software Verification. LNCS 7682*, B. Meyer and M. Nordio, Eds. Springer Berlin Heidelberg, 2012, pp. 1–30.

[13] X. Franch, L. López, C. Cares, and D. Colomer, "The *i** Framework for Goal-Oriented Modeling," in *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*, D. Karagiannis, H. C. Mayr, and J. Mylopoulos, Eds. Springer International Publishing, 2016, pp. 485–506.

[14] R. Salay, M. Chechik, J. Horkoff, and A. Di Sandro, "Managing Requirements Uncertainty with Partial Models," *J. Requirements Engineering*, vol. 18, no. 2, pp. 107–128, Jun 2013.

[15] Aprajita, S. Luthra, and G. Mussbacher, "Specifying Evolving Requirements Models with TimedURN," in *Proc. of MiSE@ICSE'17*, May 2017, pp. 26–32.

[16] G. Mathew, T. Menzies, N. Ernst, and J. Klein, ""SHORT"er Reasoning About Larger Requirements Models," in *Proc. of RE'17*, 2017, pp. 154–163.

[17] C. M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Modeling and Reasoning on Requirements Evolution with Constrained Goal Models," in *Proc. of SEFM'17*, 2017, pp. 70–86.

[18] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos, "Representing and Reasoning about Preferences in Requirements Engineering," *Requirement Engineering*, vol. 16, no. 3, pp. 227–249, Aug 2011.

[19] I. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a New Generation of Requirements Modeling Languages with Goals, Preferences, and Inconsistency Handling," in *Proc. of RE'10*, 2010, pp. 115–124.

[20] B. Zarrin and H. Baumeister, "An Integrated Framework to Specify Domain-Specific Modeling Languages," *Proc. of 6th International Conference on Model-Driven Engineering and Software Development*, pp. 83–94, 2018.

[21] S. Melnik, P. A. Bernstein, A. Halevy, and E. Rahm, "Supporting Executable Mappings in Model Management," *Proc. of SIGMOD'05*, pp. 167–168, 2005.